REMARKS

This paper is being provided in response to the October 16, 2006 Final Office Action for the above-referenced application.

The rejection of claims 1-9 under 35 U.S.C. 103(a) as being unpatentable over the article "File System Design for an NFS Server" by Hitz, et al. (hereinafter "File System Design article") in view of U.S. Patent No. 5,819,292 to Hitz et al. (hereinafter "Hitz") in view of the article "A Persistent Snapshot Device Driver for Linux" by Siddha (hereinafter "Siddha") in view of U.S. Patent No. 6,460,054 to Grummon (hereinafter "Grummon") is hereby traversed and reconsideration thereof is respectfully requested.

Claim 1 recites a method of restoring data to a first storage device. The method includes providing data in the first storage device at a first storage area of a first type that contains sections of data, providing data in a second storage device at a second storage area of a second type where the second type has, for each section of data thereof, at least one of: a pointer to a corresponding section of data of the first storage area and a pointer to corresponding section of data of a third storage device at a third storage area of the first type, wherein prior to writing new data to a section of the first storage area pointed to by a pointer of the second storage area, data of the section of the first storage area is copied to a section of the third storage area and the pointer of the second storage area is adjusted to point to the section of the third storage area, providing data in a fourth storage device having at least one other storage area of the second type, and, for each particular section of data of the second storage area having a pointer to the third storage area, providing to a corresponding section of the first storage area an indirect

pointer to a corresponding section of the third storage area if no storage areas of the at least one other storage area point to the corresponding section of the first storage area. Claims 2-9 depend from claim 1.

The File System Design article discloses a method for restoring data using a snapshot technique. Section 3.4 beginning on page 10 illustrates the snapshot technique where figure 3(a) shows the file system prior to creating the snapshot, figure 3(b) shows the file system after creating the snapshot, and figure 3(c) shows what happens when a user modifies a data block.

Hitz discloses a method for maintaining consistent states of a file system. Figures 18A-18C illustrate a snapshot operation where a snapshot inode 1822 is provided as a snapshot of the file system block 1810. Note that figures 18A-18C of Hitz are substantially similar to figures 3(a)-3(c) of the File System Design article. Similarly, the corresponding descriptions of how snapshots are created and handled in both Hitz and the File System Design article are substantially similar. Accordingly, the discussion below of Hitz is also applicable to the File System Design article.

Sidda discloses a technique for providing a snap copy of a file where pointers point to blocks of the file and, in response to a writing new data to a particular block, previous data for the block is first copied to the a location prior to allowing the write to the block for the file. See, for example, the example illustrated by Figure 1 of Siddha where a snap copy of a file called "project.txt" is made and a particular block ("Block C" in the example provided by Sidha) is copied to a new location prior to a write to Block C at its initial location.

Grummon teaches a system for management of storage devices where storage areas and data sections are defined corresponding to tracks/storage devices.

It is worth noting Hitz does not show, teach or suggest the feature recited in Applicants' independent claim 1 where, in response to a write to a section of the stored data pointed to by a pointer of the table of the virtual storage area, data is copied from the storage data to a section of another storage area prior to the write and the pointer (of the virtual storage area) is caused to point to the other storage area. This recited feature may be understood by reviewing Applicants' specification at, for example, Figure 3 which shows the virtual storage area (84) pointing to both the standard logical device (82) and the log device (86). Each of the pointers from the virtual storage device (84) to the log device (86) correspond to a write to a corresponding section of the standard logical device (82). Thus, in response to a write to a section of the standard logical device (82), the old data is first copied to the log device (86), the corresponding pointer is made to point to the log device (86), and then the write is provided to the standard logical device (82).

In contrast to the feature recited in Applicants' claim 1, discussed above, Hitz discloses that, when data is written to the file system, the data is copied to a new location (e.g., copied from the old block 1818 to the new block 1824 in Figure 18C) and the device to which the write occurred is made to point to the new data block 1824. Thus, unlike Applicants' claimed invention where the virtual device points to the moved old data, Hitz teaches the opposite where the original device to which the write is being made points to a different block that is allocated. The disadvantage of the system disclosed in Hitz is that the existence of the snapshot inode 1822 causes a change in the operation of the standard device 1810 (e.g., pointing to the block 1824

instead of the block 1818 when a write occurs). That is, if the snapshot inode 1822 of Figure 18C were not present in Hitz, the block 1810 would point to the block 1818 rather than pointing to a new block 1824. In contrast, the present claimed invention does not cause any alteration in where the data of the standard device is located because all of the changes due to writes are handled by the virtual device and log device.

Applicants respectfully submit that the addition of Siddha and/or Grummon to Hitz does not overcome the above-noted deficiencies of Hitz because, as explained in more detail below, the combination of Hitz with Siddha to render the claims obvious as suggested in the Office Action would change the principle of operation of Hitz and would require substantial reconstruction and redesign. In addition, as discussed in detail below, Hitz teaches away from using the copy-on-write technique incorporated from Siddha into Hitz to reject the claims.

Hitz discloses a WAFL file system that uses an inode table and indirect blocks to point to data blocks of the file system. Hitz also describes the use of "anodes", but further discloses that an anode table is equivalent to an inode table. Figures 18A-18C of Hitz illustrate a snapshot operation where a snapshot inode 1822 is provided as a snapshot of the file system block 1810. As described in column 19 of Hitz, figure 18C is a diagram illustrating the snapshot 1822 when a change to the active file system occurs after the snapshot 1822 is taken. Figure 18C show an instance where a block 1818 comprising data "D" is modified after the snapshot was taken (in figure 18B). A new block 1824 containing new data D' is allocated and pointed to by the file block 1810 while the snapshot block 1822 maintains its pointer to the block 1818 containing the old data, D.

Hitz indicates that the disclosed WAFL file system addresses difficulties associated with the prior art. Figure 1 of Hitz illustrates the prior art by showing an original anode (inode) 110 and an original indirect block 112. A clone is constructed by copying all of the anode blocks and all of the indirect blocks. For figure 1 of Hitz, the clone anode 120 is a copy of the original anode 110 and the indirect block 122 is a copy of the indirect block 112. Both the original anode 110 and the clone anode 120 contain direct pointers that point directly to data blocks 114. The original anode 110 and the clone anode 120 also contain pointers to the indirect blocks 112, 122 that point directly to data blocks 124, 126.

In connection with creating the clone of figure 1, a COW (Copy On Write) flag (bit) is set for each of the pointers of the original anode and of the indirect blocks to indicate that the pointer of the original file system and the pointer of the clone file system point to the same data blocks. Subsequently, when data is written to a block having a pointer thereto with the COW flag set, a new block is allocated in the file system and updated with the modification. In addition, the COW flag in the pointer is cleared so that, for subsequent writes to the same block, no special processing is performed.

The problem with the prior art of figure 1, according to Hitz, is that a clone operation involves duplicating all of the anodes and all of the indirect blocks in the file system. As the number of files increases (i.e., tends towards a system with many small files), the percentage of the total disk space in the file system used for overhead for the clones increases. Hitz purports to address this problem with his disclosed WAFL system.

In Hitz, after the snapshot and subsequent write to the block 1818 containing the data, D, the active file system comprises blocks 1812, 1814, 1816, 1820, 1824 but does not contain the block 1818 containing the data, D. However, the block 1818 containing the data D is not overwritten. Rather, the block 1818 is protected against being overwritten by a snapshot bit that is set for the entry for the block 1818. Thus, the snapshot 1822 points to the unmodified block 1818 as well as the other blocks 1812, 1814, 1816, 1820. In effect, a write to the block 1818 containing the data, D, causes the data D to be copied to a new block 1824 (unused disk space) that is pointed to by the file system block 1810 while the snapshot 1822 maintains its previous pointer to the old block 1818 (currently used disk space).

Accordingly, both the prior art and the WAFL system described by Hitz maintain a snapshot for a file system by writing new data to new locations. Moreover, Hitz teaches that the disclosed improvement over the prior art is facilitated by writing new data to new locations. Column 18, lines 24-32, of Hitz state:

> The result is a new file system tree rooted by snapshot inode N that references exactly the same disk blocks as the root inode. Setting a corresponding bit in the blkmap for each block in the snapshot prevents snapshot blocks from being freed even if the active file system no longer uses the snapshot blocks. *Because* WAFL *always* writes new data to unused disk locations, the snapshot tree does not change even though the active file system changes. (emphasis added)

In contrast, independent claim 1 recites that, in response to a write to a section of the stored data pointed to by a pointer of the table of the virtual storage area, data is copied from the storage data to a section of another storage area prior to the write and the pointer (of the virtual storage area) is caused to point to the other storage area. Nonetheless, claim 1 is rejected using the Siddha and Grummon references to overcome this deficiency of Hitz.

Siddha discloses a technique for providing a snap copy of a file where pointers point to blocks of the file and, in response to a writing new data to a particular block, previous data for the block is first copied to the a location prior to allowing the write to the block for the file. See, for example, the example illustrated by Figure 1 of Siddha where a snap copy of a file called "project.txt" is made and a particular block ("Block C" in the example provided by Siddha) is copied to a new location prior to a write to Block C at its currently used location.

It is respectfully submitted that modifying Hitz as taught by Siddha as suggested in the Office Action would change the principle of operation of Hitz. Hitz teaches that the system described therein operates because the WAFL system always writes new data to an *unused disk location* rather than to the *currently used location*. It is respectfully submitted that modifying Hitz according to Siddha as suggested in the Office Action to write new data to the *currently used location* and coping the old data to an *unused disk location* would counteract the storage efficiencies that the Hitz system is meant to provide. As mentioned above, Hitz specifically states that because WAFL *always* writes new data to unused disk locations, the snapshot tree does not change even though the active file system does. Modifying Hitz according to Siddha as suggested by the examiner would make this no longer true and accordingly, the combination of

Siddha with Hitz as suggested in the Office Action would change the principle of operation of Hitz.

In further support of this, it is noted that FIG. 10 of Hitz is a diagram illustrating a file referenced by a WAFL inode 1010. The file includes indirect WAFL buffers 1020-1024 and direct WAFL buffers 1030-1034. The WAFL incore inode 1010 includes standard inode information 1010A (including a count of dirty buffers), a WAFL buffer data structure 1010B, 16 buffer pointers 1010C and an on-disk inode 1010D. The WAFL buffer data structure 1010B is disclosed as including two pointers where the first one references the 16 buffer pointers 1010C and the second references the on-disk block numbers 1010D.

The indirect WAFL buffer 1020 is disclosed as including a WAFL buffer data structure 1020A, a buffer 1020B that includes 1024 WAFL buffer pointers and a buffer 1020C with 1024 on-disk block numbers. The WAFL buffer data structure includes two pointers. One pointer of WAFL buffer data structure 1020A references a buffer 1020B and a second pointer references buffer 1020C. In FIG. 10, the 16 buffer pointers 1010C of WAFL inode 1010 point to the 16 single-indirect WAFL buffers 1020-1024. In turn, WAFL buffer 1020 references 1024 direct WAFL buffer structures 1030-1034. WAFL buffer 1030 is representative of direct WAFL buffers. The direct WAFL buffer 1030 is disclosed as including a WAFL buffer data structure 1030A and a 4 KB direct buffer 1030B containing a cached version of a corresponding on-disk 4 KB data block.

The WAFL incore inode 1010 contains 16 buffer pointers 1010C. In turn, the 16 buffer pointers 1010C are referenced by a WAFL buffer structure 1010B that roots the tree of WAFL buffers 1020-1024 and 1030-1034. Each WAFL inode 1010 contains a WAFL buffer structure 1010B that points to the 16 buffer pointers 1010C in the inode 1010. As indicated As indicated beginning in column 8, line 2 of Hitz, this facilitates algorithms for handling trees of buffers that are implemented recursively. As also indicated near the top of column 8, if the 16 buffer pointers 1010C in the inode 1010 were not represented by a WAFL buffer structure 1010B, the recursive algorithms for operating on an entire tree of buffers 1020-1024 and 1030-1034 would be difficult to implement.

Figures 18A-18C of Hitz illustrate a snapshot operation where a snapshot inode 1822 is provided as a snapshot of the file system block 1810. As described in column 19 of Hitz, figure 18C is a diagram illustrating the snapshot 1822 when a change to the active file system occurs after the snapshot 1822 is taken. Figure 18C show an instance where a block 1818 comprising data "D" is modified after the snapshot was taken (in figure 18B). A new block 1824 containing new data D' is allocated and pointed to by the file block 1810 while the snapshot block 1822 maintains its pointer to the block 1818 containing the old data, D.

FIG.'s 18A-18C do not explicitly show any indirection like that, for example, of FIG. 10. However, column 18, lines 50-52 of Hitz state: "FIG. 18A is a diagram of the file system 1830, before a snapshot is taken, *where levels of indirection have been removed to provide a simpler overview of the WAFL file system.*" (emphasis added). As explained in detail below,

reintroducing the levels of indirection demonstrates why Hitz cannot be modified by Siddha in the manner suggested to reject the present claims.

FIG. 19 of Hitz illustrates several levels of indirection for block 1824 of FIG. 18C. The new block 1910 corresponds to the block 1824 containing D' in FIG. 18C. However, because the block 1910 containing modified data D' is referenced by double indirection in FIG. 19, two other blocks, 1918 and 1926, are also modified. The pointer 1924 of single-indirect block 1918 references new block 1910 and so block 1918 is copied, modified, and written to disk in a new location. Similarly, pointer 1928 of indirect block 1926 is copied, modified, and written to disk in a new location because the pointer 1928 points to block 1918. Thus, as disclosed by Hitz in connection with FIG. 19, since block 1910 is pointed to by indirect blocks 1926 and 1918, then when block 1910 is modified and stored in a new disk location, the corresponding direct and indirect blocks 1926 and 1918 are copied and modified and stored in a new disk location. The blocks 1910, 1918, and 1926 are assigned to the active file system.

In FIG.'s 11A-11D, Hitz discloses a mechanism for keeping track of which intermediate nodes need to be duplicated. FIG. 11A shows a blkmap file 1110 that contains a 32-bit entry 1110A-1110D for each block. FIG. 11B is a diagram of a block entry 1110A of the blkmap file. As shown in FIG. 11B, entry 1110A is comprised of 32 bits (BIT0-BIT31). Bit 0 (BIT0) of entry 1110A is the active file system bit (FS-bit), which Hitz discloses as indicating whether or not the corresponding block is part of the active file system. Hitz distinguishes between blocks that are part of the active file system (i.e., blocks that are used for active files accessed by applications)

and blocks that are not part of the active file system, but are maintained because the blocks contain snapshot data..

Hitz also discloses that bits 1-20 (BIT1-BIT20) of entry 1110A indicate whether the block is part of a corresponding snapshot. Different snapshots are numbered 1-20 so that snapshot 1 corresponds to BIT1, snapshot 2 corresponds to BIT2, etc. Having at least one snapshot bit set means that the block is being maintained for the at least one snapshot and may not be overwritten.

Hitz discloses that when bit 0 (BIT0), also referred to as the FS-bit, is set to a value of 1, the entry 1110A of blkmap file 1110 indicates a block that is part of the active file system. Hitz also discloses that if bit 0 (BIT0) is set to a value of 0, this does not necessarily indicate that the block is available for allocation. All the snapshot bits must also be zero for the block to be available for allocation. Thus, a block having an FS-bit of zero but at least one of the snapshot bits set is being used by the corresponding at least one snapshot.

It is respectfully submitted that implementing the combination of Hitz and Siddha suggested in the Office Action would require substantial reconstruction and redesign of Hitz for a number of reasons. First, there is no disclosure in Hitz, Siddha, or any of the other cited references to perform the specific steps to transition to a state where some, but not all, of the indirect inodes are duplicated. Second, it's not clear how to set the snapshot bit for an affected indirect inode in a way to maintain smooth operation of the system following transition into the state suggested by the combination. If the snapshot bit is maintained as set to 1 for affected

indirect inodes after copying some, but not all, of the blocks to which the indirect inode points, then subsequent writes involving the indirect inode will cause the indirect inode to be incorrectly duplicated again, since Hitz discloses that a block having at least one snapshot bit set to 1 may not be overwritten. On the other hand, if the snapshot bit for the affected indirect inode is set to 0 after copying some, but not all, of the blocks to which the indirect inode points, then there is no mechanism to appropriately handle subsequent writes to the block which have not yet been copied.

Thus, Applicants respectfully submit that the combination of Hitz and Siddha suggested by the Examiner would change the principle of operation of the Hitz system *and* would require substantial reconstruction and redesign and thus may not be used to reject Appellants' claims.

In addition, Column 22, lines 4-7 of Hitz provide: "The present invention limits the total number of snapshots and keeps a blkmap file that has entries with multiple bits for tracking the snapshots *instead of using pointers having a COW bit as in Episode*." (emphasis added). As described in the background section of Hitz (see FIG.1), a COW (Copy On Write) flag (bit) is set for each of the pointers of the original anode and of the indirect blocks to indicate that the pointer of the original file system and the pointer of the clone file system point to the same data blocks. Subsequently, when data is written to a block having a pointer thereto with the COW flag set, a new block is allocated in the file system and updated with the modification. In addition, the COW flag in the pointer is cleared so that, for subsequent writes to the same block, no special processing is performed.

Thus, the COW flag mechanism disclosed in the background section of Hitz is the same mechanism borrowed from Siddha and combined with the disclosure of Hitz to reject Appellants' claims. Hitz specifically teaches against using this mechanism as set forth in the above-quoted column 22, lines 4-7 of Hitz.

Furthermore, it is not surprising that Hitz teaches against using the copy-on-write technique of Siddha since, as discussed in detail above, it is difficult to appropriately handle intermediate (indirect) inodes using Siddha's copy-on-write technique combined with Hitz' feature where each snapshot requires only a single node be created, and thereafter requires the duplication of only those data blocks that have been modified. This is perhaps why, although FIG. 1 of Hitz and the corresponding description (prior art) set forth a copy-on-write technique like Siddha, Hitz uses an entirely different technique when describing his system that minimizes duplication of inodes.

Based on the above, Applicants respectfully request that this rejection be withdrawn.

The rejection of claims 10-16 under 35 U.S.C. 103(a) as being unpatentable over the File System Design article in view of Hitz in view of Siddha is hereby traversed and reconsideration thereof is respectfully requested.

Claim 10 recites computer software, provided in a computer-readable storage medium, that restores data to a first storage area of a first type that contains sections of data from a second storage area of a second type that has, for each section of data thereof, at least one of: a pointer to

a corresponding section of data of the first storage area and a pointer to corresponding section of data of a third storage area of the first type where there is at least one other storage area of the second type. The software includes executable code that, prior to writing new data to a section of the first storage area pointed to by a pointer of the second storage area, copies data of the section of the first storage area to a section of the third storage area and adjusts the pointer of the second storage area to point to the section of the third storage area, executable code that iterates through each section of the second storage area, and executable code that provides to a corresponding section of the first storage area an indirect pointer to a corresponding section of the third storage area if no storage areas of the at least one other storage area point to the corresponding section of the first storage area. Claims 11-16 depend from claim 10.

Claim 10 recites features similar to those of claim 1, discussed above. Furthermore, the rejection of claim 10 is based on prior art references used to reject claim 1. Accordingly, Applicants respectfully submit that claim 10, and claims which depend therefrom, are patentable over the cited prior art for reasons similar to those discussed above in connection with the rejection of claim 1. Accordingly, Applicants respectfully request that this rejection be reconsidered and withdrawn.

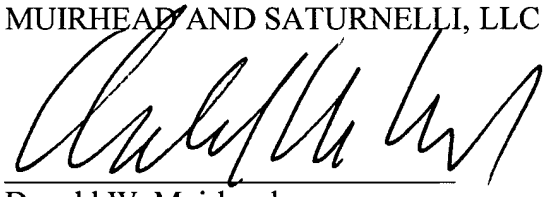
The rejection of claims 17 and 18 under 35 U.S.C. 103(a) as being unpatentable over the File System Design article in view of Hitz in view of Siddha in view of Grummon is hereby traversed and reconsideration thereof is respectfully requested.

Claims 17 and 18 depend from claim 10, discussed above. The cited references are also discussed above.

Applicants respectfully submit that the deficiencies of the File System Design article, Hitz, and Siddha with respect to claim 10, discussed above, are not overcome by the addition of the Grummon reference. Accordingly, Applicants respectfully request that this rejection be withdrawn.

Based on the above, Applicants respectfully request that the Examiner reconsider and withdraw all outstanding rejections and objections. Favorable consideration and allowance are earnestly solicited. Should there be any questions after reviewing this paper, the Examiner is invited to contact the undersigned at 508-898-8603.

Respectfully submitted,
MUIRHEAD AND SATURNELLI, LLC



December 22, 2006
Date

Donald W. Muirhead
Registration No. 33,978

Muirhead and Saturnelli, LLC
200 Friberg Parkway, Suite 1001
Westborough, MA 01581
508-898-8601 (main)
508-898-8602 (fax)
Customer No. 52427